Modelo de regresión para la predicción de precios de automóviles

Data science essential training

DiplomadosOnline.com

Ejecute las siguientes lineas para continuar.

In [1]:

```
import pandas as pd
import numpy as np

import seaborn as sns
from matplotlib import pyplot as plt
```

In [2]:

```
df = pd.read_csv('data.csv')
```

In [3]:

```
df.columns = df.columns.str.lower().str.replace(' ', '_')

string_columns = list(df.dtypes[df.dtypes == 'object'].index)

for col in string_columns:
    df[col] = df[col].str.lower().str.replace(' ', '_')
```

In [4]:

```
np.random.seed(2)

n = len(df)

n_val = int(0.2 * n)
n_test = int(0.2 * n)
n_train = n - (n_val + n_test)

idx = np.arange(n)
np.random.shuffle(idx)

df_shuffled = df.iloc[idx]

df_train = df_shuffled.iloc[:n_train].copy()
df_val = df_shuffled.iloc[n_train:n_train+n_val].copy()
df_test = df_shuffled.iloc[n_train+n_val:].copy()
```

In [5]:

```
y_train_orig = df_train.msrp.values
y_val_orig = df_val.msrp.values
y_test_orig = df_test.msrp.values

y_train = np.log1p(df_train.msrp.values)
y_val = np.log1p(df_val.msrp.values)
y_test = np.log1p(df_test.msrp.values)

del df_train['msrp']
del df_val['msrp']
del df_test['msrp']
```

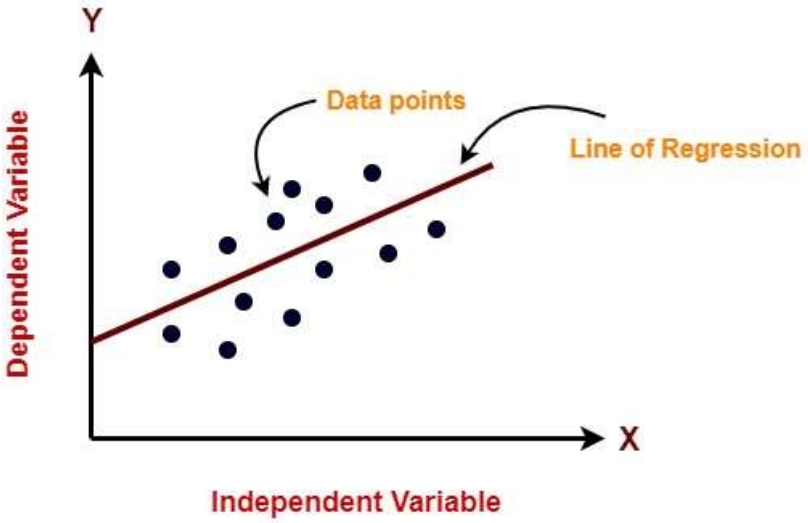
2) Modelado de los datos

Tras realizar el análisis inicial de los datos, estamos preparados para entrenar un modelo. El problema que estamos resolviendo es un problema de regresión: el objetivo es predecir un número, el precio de un automovil.

Para este proyecto utilizaremos el modelo de regresión más sencillo: la regresión lineal.

2.1 Regresión lineal

En primer lugar, repasemos cómo funciona la regresión lineal.



Supongamos que tenemos una observación simple x_i y el valor y i que queremos predecir. El índice i significa aquí que se trata de la observación número i, una de las m observaciones que tenemos en nuestro conjunto de datos de entrenamiento.

Entonces, para esta única observación, el modelo a estimar tiene la forma:

$y_i \approx f(x_i)$

Si tenemos n características, nuestro vector x_i sería $n-$dimensional, por lo que tiene n componentes:

$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$

Como tiene n componentes, podemos escribir la función f como una función con n parámetros, que es lo mismo que la fórmula anterior:

$y_i \approx f(x_i) = f(x_{i1}, x_{i2}, \ldots, x_{in})$

En nuestro caso, tenemos $7,150$ autos en el conjunto de datos de entrenamiento. Esto significa que $m = 7,150$ y i puede ser cualquier número entre 0 y 7,149.

Si f es el modelo de regresión lineal, tiene la siguiente forma:

$f(x_i) = f(x_{i1}, x_{i2}, \ldots, x_{in}) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}$

donde, $\beta_0, \ldots, \beta_n$ son los parámetros del modelo:

- β_0 es el término de sesgo
- $\beta_1, \ldots, \beta_n$ son las ponderaciones de cada característica $x_{i1}, x_{i2}, \ldots, x_{in}$.

Estos parámetros definen exactamente cómo debe combinar el modelo las características para que las predicciones al final sean lo mejor posible.

Para que la fórmula sea más corta, vamos a utilizar la notación de suma:

$f(x_i) = f(x_{i1}, x_{i2}, \ldots, x_{in}) = \beta_0 + \sum_{j=1}^{n} \beta_j x_{ij}$

Entrenamiento:

¿Cuáles son x_i e y_i para este problema?

Estas ponderaciones son las que aprende el modelo cuando lo entrenamos.

Este modelo lo podemos escribir es forma matricial como:

$f(X) = \beta_0 + X\beta,$

donde, β es el vector columna de los coeficientes $\beta_1, \beta_2, \ldots, \beta_n$, mientras X es lo que se conoce como la matiz de diseño que contiene todas las observaciones de las características consideradas.

2.2 Entrenamiento del modelo de regresión

Para poder hacer prediciones, necesitamos saber las ponderaciones β. ¿Cómo las obtenemos?

Aprendemos los pesos a partir de los datos: utilizamos la variable objetivo y para encontrar la beta que combina las características de X de la mejor manera posible.

En el caso de la regresión lineal, “la mejor manera posible ”significa que minimiza el error entre las predicciones $f(X)$ y el objetivo real y.

Tenemos varias formas de hacerlo. Utilizaremos la ecuación normal, que es el método más sencillo. El vector de pesos β se puede calcular con la siguiente fórmula:

$$\beta = (X^T X)^{-1} X^T y$$

Esto es fácil de traducir a NumPy:

- X^T es la transpuesta de X. En NumPy, es `X.T`
- $X^T X$ es una multiplicación de matrices, que podemos hacer con el método del punto de NumPy: `X.T.dot(X)`
- X^{-1} es la inversa de X. Podemos utilizar la función `np.linalg.inv` para calcular la inversa.

Para implementar la ecuación normal, tenemos que hacer lo siguiente:

- Crear una función que tome una matriz X con características y un vector y con el objetivo.
- Añadir una columna ficticia (la característica que siempre se pone en 1) a la matriz X.
- Entrena el modelo: calcula los pesos β mediante la ecuación normal.
- Dividir este β en el sesgo β_0 y el resto de los pesos, y devolverlos.
- El último paso (dividir β en el término de sesgo y el resto) es opcional y se hace principalmente por conveniencia.

La implementación sería:

In [8]:

```
def train_linear_regression(X, y):
    ones = np.ones(X.shape[0])
    X = np.column_stack([ones, X])

    XTX = X.T.dot(X)
    XTX_inv = np.linalg.inv(XTX)
    w = XTX_inv.dot(X.T).dot(y)

    return w[0], w[1:]
```

Entrenamiento:

Explique cada linea de la función e identifíquela con los pasos señalados.

In [ ]:

```

```